PROGRAMMING LANGUAGES — A BRIEF REVIEW

F. Marsing

Translation of: "Programmiersprachen - Eine kurzgefasste Übersicht," Elektronik, Vol. 21, June 1972, pp. 213 - 216.

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
WASHINGTON, D. C. 20546 OCTOBER 1972

12 P8

PROGRAMMING LANGUAGES — A BRIEF REVIEW

F. Marsing

ABSTRACT. Brief description of the most important and most commonly used programming languages and their ranges of applicability. The advantages of programming into a special programming language and then into computer language rather than directly into computer language are illustrated. Brief descriptions are given of system-dependent and problem-oriented (or system-independent) programming languages. In the case of the problem-oriented languages a distinction is made between programming languages for commercial problems and programming languages for mathematical and technical problems. Figuring in the latter group are the well-known ALGOL and FORTRAN programming languages.

In addition to other factors, the selection of the programming language(s) /213* is of decisive importance for the effectiveness of a data processing system. The available software offered by manufacturers of data processing installations includes a large group of language translators for programming languages, which are characterized by increasing ease of use and efficiency. Various programming languages are available which are oriented towards solving problems usually treated with data processing. Here we will give a short description of the most important and most used programming languages and their range of application.

1. MACHINE LANGUAGE

Any data processing installation has a fixed set of instructions (commands) which it can carry out in order to solve a given problem. These instructions

*Numbers in the margin indicate pagination in the original foreign text.

must be written in a special binary code, the machine code. All programs to be executed must therefore be converted into machine code or machine language. It is the task of the language translators to translate programs available in a certain programming language into machine language.

Why are programming languages used at all and why are the programs not directly written in machine language? Figure 1 shows an instruction written in machine language using sedecimal coding. This instruction causes the content of the working storage locations 90 and 91 to be transferred into storage locations 80 and 81. This representation immediately shows the disadvantages of programming in machine language:

1. It is very difficult to learn the code of a machine language. For each of the numerous instructions, the programmer must also learn the corresponding abstract numerical code.

2. The programmer must count off the storage addresses to determine where the information being processed is located. Only then can these addresses be substituted in the corresponding instructions.

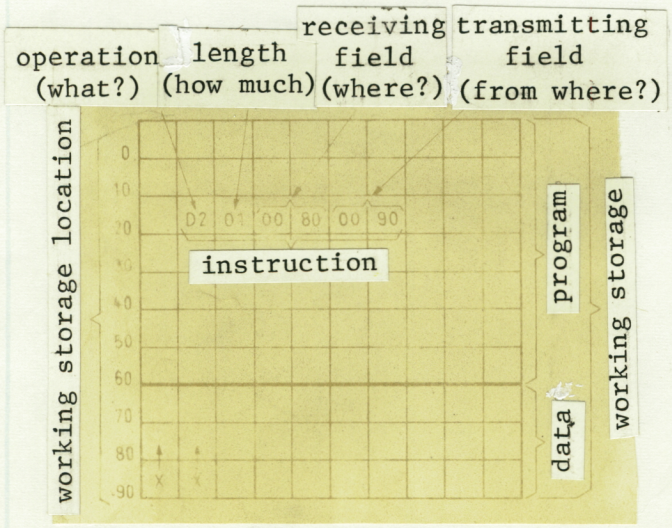

Fig. 1. Instruction in machine code (sedecimal representation).

3. Programs available in machine language are very difficult to read and hard to interpret. This is a particular disadvantage if already existing programs are to be improved or changed.

These disadvantages led to programming languages very early in the history of data processing. Instead of the abstract machine codes, there are mnemonic coded expressions or expressions used in conversational (English) language. Instead of absolute addresses, there are symbolically addressed fields and regions. The instruction shown in Figure 1 has the following appearance when such a programming language is used:

Operation	Length	Receiving Field	Transmitting Field	Machine Language
D2	01	0080	0090	
MVC		B	A	
MOOVE CHARACTERS (transfer characters)		Receiving Field	Transmitting Field	Programming Language

The language translator converts the mnemonic language code into machine code and replaces the symbolic addresses A and B by the absolute addresses which it has determined. It determines the number of digits to be transferred from the defined length of the fields A and B. The programs written in programming language are easier to write, to interpret and to change than programs written in machine language.

Two directions were followed in the development of these programming languages. One of these lead to the equipment-oriented programming languages, the other to the problem-oriented programming languages. By equipment-oriented we mean that the language refers to the specific instruction center of a certain installation and that to each available machine instruction there is a corresponding instruction in the equipment-oriented programming language. On the other hand, the problem-oriented programming languages are not conceived for special data processing installations. They were instead developed for the solution of commercial or mathematical problems. Therefore, they are also called equipment-independent programming languages.

2. EQUIPMENT DEPENDENT PROGRAMMING LANGUAGES

Equipment dependent programming languages are called assembler languages. Corresponding translators are called assemblers. Assembler languages were the direct consequence of the machine language of a certain data processing installation. The binary operational coding of the machine language was transformed into a mnemonic coding. The absolute data addresses in the instructions were replaced by symbolic addresses. It is the task of the translator to produce a machine program which can run from an available program written in assembler language, the primary program. The translation ratio is 1:1. The assembler produces one machine instruction from one instruction of the primary program.

In this way, the programmer influences the detailed configuration of the machine program and can design it in an optimum way. It should be noted that as a rule, an optimum can either be produced with respect to storage requirements or running time of the program. Because of the translation ratio of 1:1, the error rate in larger programs written in assembler language is greater for the formal errors as well as logical errors compared with the problem-oriented programming languages. This also means that test time is greater.

The assembler language consists of the following components:

2.1 Instructions

Instructions are directives to the installation to carry out certain operations with certain data. Instructions are available for the following:

Input and output of data
Transfer of data
Decimal arithmetic
Fixed point arithmetic
Floating point arithmetic
Comparison
Decisions
Logical data processing

2.2 Allocations

Areas and fields are reserved in the program by means of allocations. These are used for input, storage and processing of data.

Values required by the program can also be defined by means of allocations, unless they are read in from external data sources.

2.3 Macros

Macros are program building blocks. Within them, parts of programs and instruction sequences are stored which appear often in the program in the same or only slightly modified form. A program does not become shorter when macros are used, but the programming effort is reduced.

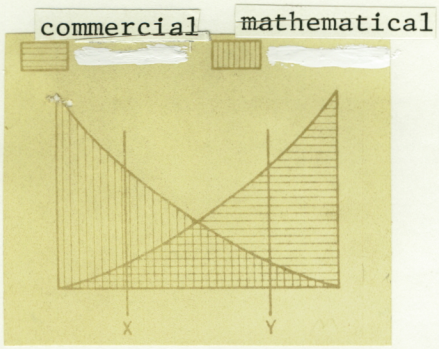

Fig. 2. Distribution of commercial and mathematical components in data processing program.

2.4 Comments

Comments are used to identify program steps with comments. This makes a program clearer and subsequent users can understand it more readily.

3. PROBLEM ORIENTED (EQUIPMENT-INDEPENDENT) PROGRAMMING LANGUAGES

Two large characteristic application areas exist in data processing. One is the area of commercial applications. The other include the mathematical-technical applications. Simplifying we may say that in the commercial applications (for example, payroll calculations) large amounts of data are processed using a relatively uncomplicated processing method. The data input and output requires the most amount of time. For mathematical applications (for example, solution of a system of equations), the amount of input and output data is small but the internal processing steps require the greatest amount of time. This is why these are also called "input-output-intensive" and "computationally intensive" programs.

Mixed versions of the above are quite frequent in practice. Many programs consist of input-output intensive and computation-intensive program parts. Figure 2 shows the distribution of commercial and mathematical applications for present day extensive programs. Most of them lie between the values x and y shown in Figure 2.

Because of the basically different problems attacked with commercial and mathematical processing of data, there are problem oriented programming languages for commercial problems as well as for mathematical-technical problems. In both cases, they have been designed by technical panels for the requirements. They have been standardized and the instruction spectrum of certain data processing installations was not taken into account.

Any installation which is to process programs written in problem-oriented programming languages must have a translater, called a compiler. It translates the instructions of the newly written programming language into the machine code of the installation in question. The translation ratio from instructions in

the primary language to instructions in the machine language is not always 1:1 as for the assembler. It is 1:n, where n depends on the programming language being considered and on the instruction format. This means that the programmer no longer can influence the exact instruction sequence in the machine program and can no longer optimally design the program for storage requirement or program running time. Nevertheless, the instruction sequences produced by a single instruction are formulated so that there is an equal weighting of running time and storage requirements. In addition, the error rates for form errors and logical errors, test time and time requirements for learning the languages are noticeably smaller than for assembler languages.

The fact that commercial as well as mathematical-technical problems can occur in the same program are dealt with as follows in practice:

1. Commercial languages are extended to include mathematical components. Mathematical languages are extended in the commercial direction (input-output). /215 This does not always correspond to the initial language concept but has many advantages in many cases.

2. The program is divided into a commercial and a mathematical partial program. The partial programs are written in the suitable but different languages. They are then translated by the corresponding language translators. The partial programs produced by the translators, called modules, have a unified format and can be connected to a total program which can run (Figure 3) using a specially conceived program (binder).

3.1 Problem Oriented Programming Languages for Commercial Problems

a) COBOL

Cobol is the abbreviation for common business oriented language. This programming language produces an effective coding of problems from the commercial area. According to the criteria for commercial applications, cobol stresses the processing of extensive amounts of data which can come from various sources. This concept of course does not exclude the use of cobol for other than commercial applications. However, Cobol will not represent an optimum solution for them. A Cobol program must always be organized into four parts (divisions), which must

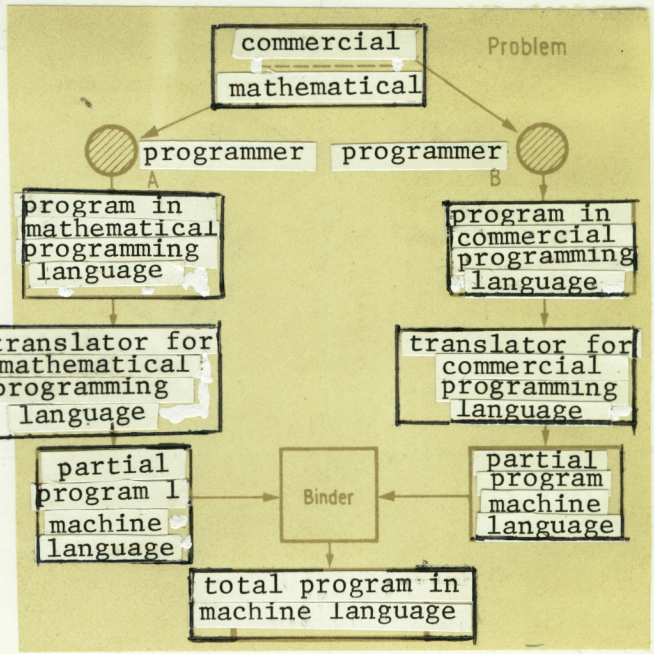

Fig. 3. Treatment of a mixed problem with various language translators.

occur in exactly the following sequence:

1. Identification Part (Identification Division)

The identification part contains information regarding: programming identification — author — data. It only has a documentary nature and does not influence the translating or running of the program.

2. Machine Part (Environment Division)

The machine part of the Cobol program contains data on: type of the computer used for translating — type of computer for program run — equipment assignment for required data.

3. Data Part (Data Division)

The data part describes the following program components: data used — structure of the data sets — input ranges — working ranges — constants.

4. Processing Part (Procedure Division)

The procedure part contains instructions which must be carried out when the program is run.

The Cobol compiler translates a program written in Cobol into machine language. During the translation errors are recognized and displayed. These could disturb an orderly running of the program.

b) The List Program Generator (LPG)

The list program generator is a special case of translator for problem oriented program languages. It does not produce several instructions in machine code from an instruction in the primary program. Instead, it selects program parts required for the problem from an available total program having a standard logic program, based on instructions from the primary program. As the name suggests, the LPG produces programs which are primarily used to print out prepared lists of data available in a data storage.

In this program, the programmer give indications of the following: selection of the input data — execution of group checks — processing (sum formation, etc.) — preparation of data — formating of output lists (paper advance, etc.).

Since it is not required for the programmer to be familiar with the special properties of the installation on which his program is running, the LPG language is called a problem-oriented programming language. The programming effort and time required to learn the LPG language are small. Nevertheless, because the standard logic of the LPG must be used, the produced programs require a great deal of storage.

3.2 Problem Oriented Programming Languages for Mathematical-Technical Problems

a) ALGOL

The programming language Algol is primarily used in Europe. It is an algorithmic formula language (algorithmic language), which is especially well suited for translation by a compiler. It allows the programmer to formulate mathematical problems in a familiar mathematical notation.

Certain functions required for mathematical computations, such as, for example, the determination of logarithms, trigonometric functions, determination of square roots, Boolian algebra, are already available in completely programmed form. They only have to be called by the programmer. If Algol is used, the user does not have to perform the detailed programming of extensive mathematical calculations. He can directly formulate his problems considering the formal rules of Algol. These are formulated as mathematical formulas, expressions, or equations. Initially, Algol did not have a way of processing extensive input and output data. In the mean time, many manufacturers have extended Algol in this direction and have made available the corresponding translators. Algol is easy and can be learned quickly. /216 This is why it is taught in many German schools.

b) FORTRAN

Fortran (formula translation) is a problem oriented programming language which like algol, is primarily used for solving mathematical and technical problems. By extending the possibilities of Fortran in the direction of commercial data processing, it is possible to formulate and solve other than mathematical problems using this language. Since the language size of Fortran is continuously being expanded, there are a large number of Fortran versions. The last and completed version (called FORTRAN IV) was standardized by the United States of America Standard Institute.

The Fastfortran compiler is one of the newest versions of Fortran translators. Fastfortran has about the same language size as Fortran. The most important difference between the two translators is the fact that the normal Fortran compiler produces a program which is written into the library of the operational system. The Fastfortran compiler is of the compile and go type and produces only a temporary machine code. The fact that there is no translation of the Fortran instruction in the case of Fastfortran but only an interpretation, and the fact that the extensive library management is dropped, makes the Fastfortran compiler ten times as fast as the normal Fortran compiler. However, the resulting programs are slower and require more storage.

4. NEW DEVELOPMENTS

A large number of new languages and new language groups have been produced

during the extension of data processing into new and wider areas. This has been done for both general purpose languages and special languages.

TABLE OF CHARACTERISTIC PROPERTIES OF SEVERAL PROGRAMMING LANGUAGES

	Assembler	COBOL	LPG	ALGOL	FORTRAN
Range of application	commercial mathematical-technical	commercial	commercial	mathematical	mathematical
Storage requirement of the machine program	optimal	average	large	large	large
Running time of the machine program	optimal	almost optimum for input-output intensive programs	slow	moderately fast	fast
Writing effort for production	very large	large	minimal	small	small
Time requirement for learning the language	large (about 4 weeks)	average (2 weeks)	small (1 week)	average (1 to 2 weeks)	average (1 to 2 weeks)

4.1 PL 1

PL 1 was developed by the firm IBM (programming language 1) and merges elements of the Cobol, Algol, and Fortran languages. This means that PL 1 is a universally applicable programming language which can be used for commercial as well as for mathematical and mixed problems without any restrictions. However, the language size of PL 1 is very large, so that more time is required to learn the language than for other higher programming languages.

4.2 Dialog Languages

Dialog languages should only be used for time sharing operation. For all the programs mentioned above, it was necessary to input the entire program on punched cards or magnetic tape. It then had to be translated. If a dialog language is used, the user can input every instruction separately using a terminal. Each instruction is immediately tested for accuracy after it is input and the

determined errors are communicated to the user. Erroneous instructions can at any time be corrected from the terminal during or after translation of the entire program.

5. CRITERIA FOR SELECTION OF A PROGRAMMING LANGUAGE

The table above gives some of the criteria used to select a programming language. The weighting of these criteria will be different in each case. It depends on: storage capacity — equipment — problem — program volume as well as load on the installation. Therefore, it cannot be solved in general terms but only for each case.

REFERENCES

1. Mrachacs, H.-P. and G. Peetz. Taschenbuch für Programmierer (Handbook for Programmers). Verlag Moderne Industrie, Munich, 1971.

2. Komarnicki, O. Programmiermethodik (Programming Methods). Springer-Verlag, Berlin-Heidelberg, 1971.

3. Löbel, G., P. Müller and H. Schmid. Lexikon der Datenverarbeitung (Lexicon of Data Processing). Verlag Siemens AG, Munich, 1969.

4. Schrader, K.-H. Über die Benutzung problemorientierter Sprachen (The Use of Problem Oriented Languages). Elektronische Datenverarbeitung, 1968, Vol. 8, pp. 400 - 406.

Translated for National Aeronautics and Space Administration under contract No. NASw 2035, by SCITRAN, P. O. Box 5456, Santa Barbara, California, 93108.